Assume that scalar-real y and two-dimensional real vector $\mathbf{x}$ are related to each other according to $y = c(\mathbf{x}, \mathbf{w}) + v$, where $c(., \mathbf{w})$ is a cubic polynomial in $\mathbf{x}$ with coefficients $\mathbf{w}$ and v is a random Gaussian random scalar with mean zero and σ^2-variance.

Given a dataset $D = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ with N samples of $(\mathbf{x}, y)$ pairs, with the assumption that these samples are independent and identically distributed according to the model, derive two estimators for $\mathbf{w}$ using maximum-likelihood (ML) and maximum-a-posteriori (MAP) parameter estimation approaches as a function of these data samples. For the MAP estimator, assume that $\mathbf{w}$ has a zero-mean Gaussian prior with covariance matrix $\gamma\mathbf{I}$.

Having derived the estimator expressions, implement them in code and apply to the dataset generated by the attached Matlab script. Using the *training dataset*, obtain the ML estimator and the MAP estimator for a variety of γ values ranging from 10^{-m} to 10^{n}. Evaluate each *trained* model by calculating the average-squared error between the y values in the *validation samples* and model estimates of these using $c(., \mathbf{w}_{trained})$. How does your MAP-trained model perform on the validation set as γ is varied? How is the MAP estimate related to the ML estimate? Describe your experiments, visualize and quantify your analyses (e.g. average squared error on validation dataset as a function of hyperparameter γ) with data from these experiments.

Note: Point split will be 20% for ML and 20% for MAP estimator results and discussion.

*ML - Estimator :→

$$\hat{\theta}_{ML} = \underset{\theta}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} \ln \frac{(P(x_i, y_i | \theta)}{P(y_i | x_i, \theta) \cdot P(x_i|\theta)}$$

Assuming x is independant of θ,

$$= \underset{\theta}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} \ln g(y_i | f(x_i,\theta), \Sigma)$$

$$= \underset{\theta}{\operatorname{argmin}} -\frac{1}{N}\sum_{i=1}^{N} \ln \left[(2\pi)^{-m/2} |\Sigma| \right]^{-1/2} \cdot e^{-1/2 (y - f(x_i,\theta))^T \Sigma^{-1} (y_i - f(x_i,\theta))}$$

$$= \underset{\theta}{\operatorname{argmin}} -\frac{1}{2N\sigma^{-2}}\sum_{i=1}^{N} (y - f_i(x_i,\theta))^T (y_i - f(x_i,\theta))$$

$$\Sigma = \sigma^2 I$$

$$f(x,\theta) = z^T\theta$$

$$\theta_{ML} = \underset{\theta}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} (y_i - z_i^T\theta)^T (y_i - z_i^T\theta)$$

Taking $\frac{d}{d\theta^T}$ on B.S we get.

$$0 = \frac{1}{N}\sum_{i=1}^{N} \left[2(y_i - z_i^T\theta) z_i \right]$$

$$\Rightarrow \frac{1}{N}\sum_{i=1}^{N} y_i z_i - \left(\frac{1}{N}\sum_{i=1}^{N} z_i z_i^T\right)\theta = 0$$

$$\Rightarrow \theta = \left(\frac{1}{N}\sum_{i=1}^{N} y_i z_i\right)\left(\frac{1}{N}\sum_{i=1}^{N} z_i z_i^T\right)^{-1}.$$

* MAP - Estimator :→

* $\theta_{MAP} = \arg\min_{\theta} -\ln\left[P(\theta|x,y)\right]$

* using Baye's theorem we get.

$$= \arg\min_{\theta} -\ln \frac{P(y|x,\theta)\,P(\theta)}{P(y|x)}$$

* $P(y|x)$ is independant of θ.

$$\Rightarrow \theta_{MAP} = \arg\min_{\theta} -\ln P(y|x,\theta) - \ln(P(\theta))$$

* $P(\theta) = \mathcal{N}(0, \gamma I)$

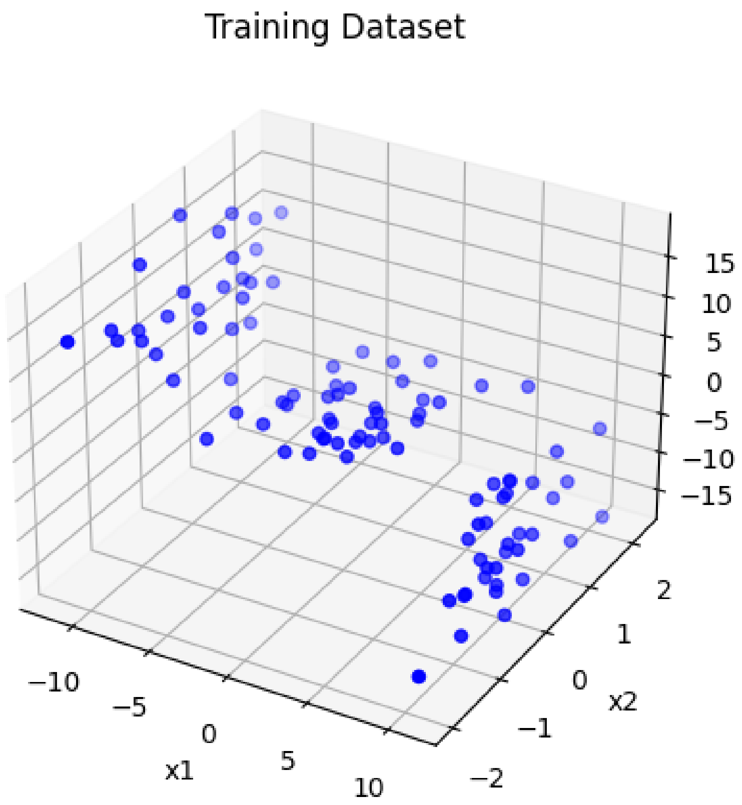
$$\Rightarrow \theta_{MAP} = \arg\min_{\theta} -\frac{1}{2\sigma^2 N}\sum_{i=1}^{N}(y_i - z_i^T\theta)^T(y_i - z_i^T\theta) - \frac{1}{2\gamma}\theta^T\theta$$

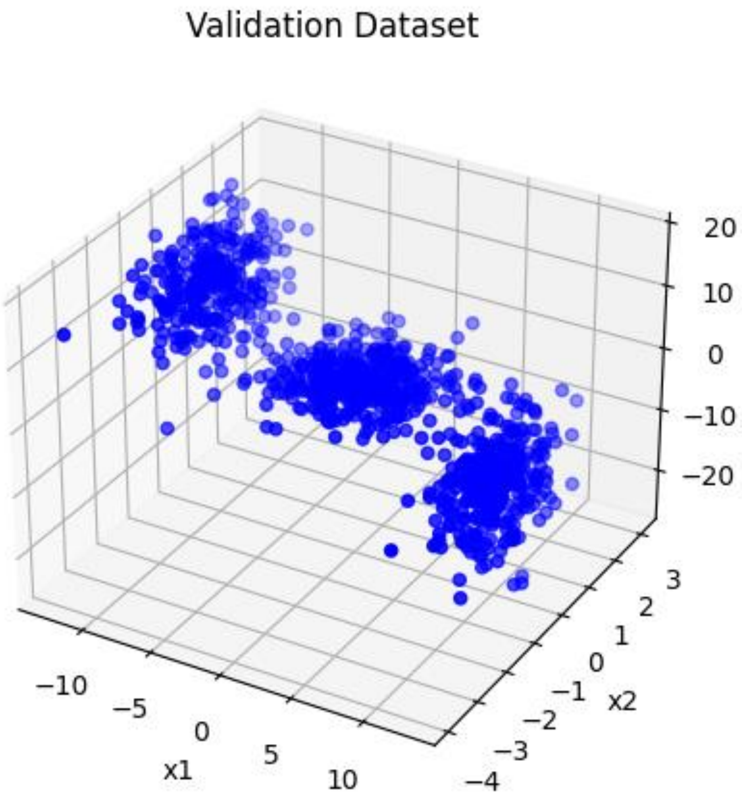
Taking $\frac{d}{d\theta}$ on B.S we get.

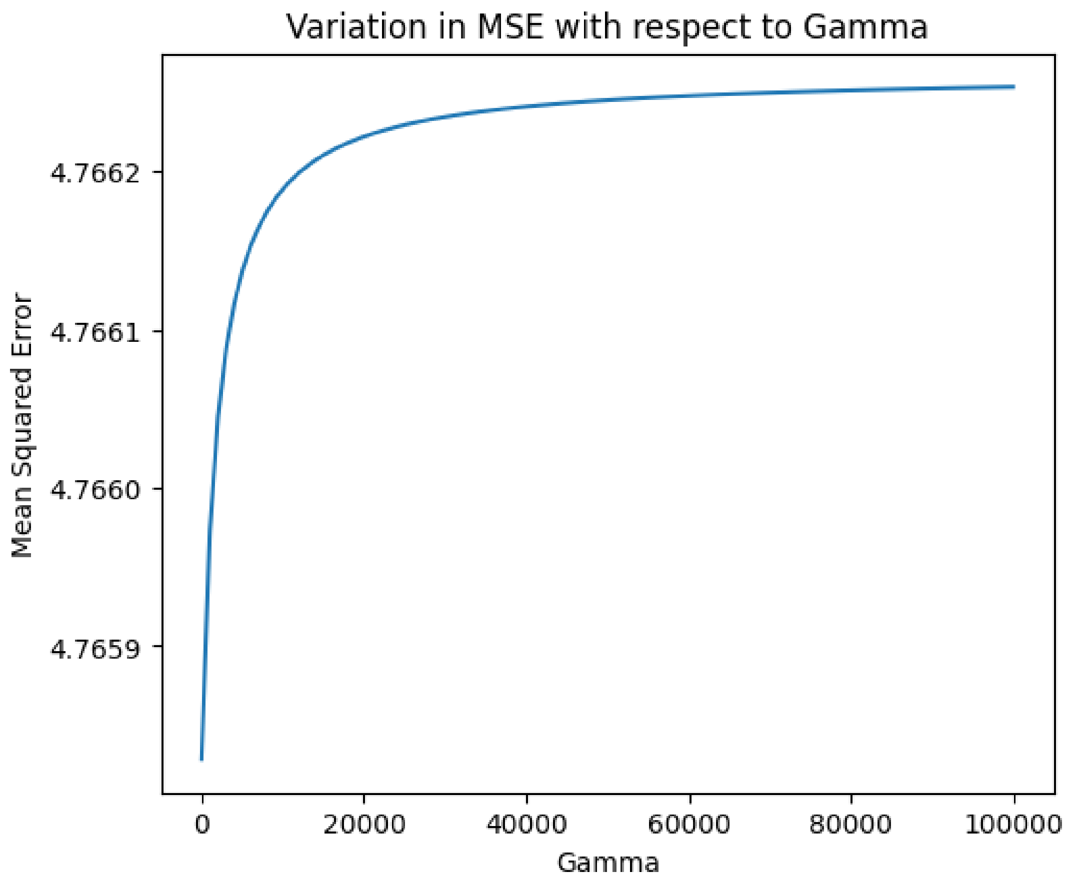
$$\Rightarrow \frac{1}{2N\sigma^2}\sum_{i=1}^{N}\left[2(y_i - z_i^T\theta).z_i\right] - \frac{1}{\gamma}\theta = 0.$$

$$\Rightarrow \left(\frac{1}{\gamma} + \frac{1}{\sigma^2 N}\sum_{i=1}^{N} z_i^T.z_i\right)\theta = \frac{1}{N\sigma^2}\sum_{i=1}^{N} y_i z_i$$

$$\Rightarrow \theta = \left(\frac{1}{N}\sum_{i=1}^{N} y_i z_i\right)\left(\frac{1}{N}\sum_{i=1}^{N} z_i^T z_i + \frac{\sigma^2}{\gamma}I\right)^{-1}$$


Training Dataset
−10
−5
0
5
10
x1
−2
−1
0
1
2
x2
15
10
5
0
−5
−10
−15


Validation Dataset
−10
−5
0
5
10
x1
−4
−3
−2
−1
0
1
2
3
x2
20
10
0
−10
−20



Maximum Mean Squared Error (MSE) Value is - 4.21747431177398.
Minimum Value of MSE is: 4.215867817598634, Corresponding Gamma Value is: 0.01.
Maximum Value of MSE is: 4.217442101245311, Corresponding Gamma Value is: 100000.0.

- The Mean Squared Error is minimum for the lowest value of γ and rises monotonically with increase in γ.
- This shows that the prior probability of w added as a regularization term in the MAP estimate has a positive impact on model performance.
- Mathematically, if γ tends to ∞ in the derived equation for MAP estimate, then it is equal to ML estimate. This can be supported by the above graph and numerical results that as γ escalates.

Observation: The MSE for MAP based model tends to approach the MSE for ML based model if γ tends to infinity.

Appendix:

```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def hw2q2():
    Ntrain = 100
    data = generateData(Ntrain)
    plot3(data[0,:],data[1,:],data[2,:], 'Training')
    xTrain = data[0:2,:]
    yTrain = data[2,:]

    NValidate = 1000
    data = generateData(NValidate)
    #plot3(data[0,:],data[1,:],data[2,:], 'Validate')
    xValidate = data[0:2,:]
    yValidate = data[2,:]

    return xTrain,yTrain,xValidate,yValidate

def generateData(N):
    gmmParameters = {}
    gmmParameters['priors'] = [.3,.4,.3] # priors should be a row vector
    gmmParameters['meanVectors'] = np.array([[-10, 0, 10], [0, 0, 0], [10, 0, -10]])
    gmmParameters['covMatrices'] = np.zeros((3, 3, 3))
    gmmParameters['covMatrices'][:,:,0] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    gmmParameters['covMatrices'][:,:,1] = np.array([[8, 0, 0], [0, .5, 0], [0, 0, .5]])
    gmmParameters['covMatrices'][:,:,2] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    x,labels = generateDataFromGMM(N,gmmParameters)
    return x

def generateDataFromGMM(N,gmmParameters):
#    Generates N vector samples from the specified mixture of Gaussians
#    Returns samples and their component labels
#    Data dimensionality is determined by the size of mu/Sigma parameters
    priors = gmmParameters['priors'] # priors should be a row vector
    meanVectors = gmmParameters['meanVectors']
    covMatrices = gmmParameters['covMatrices']
    n = meanVectors.shape[0] # Data dimensionality
    C = len(priors) # Number of components
    x = np.zeros((n,N))
    labels = np.zeros((1,N))
    # Decide randomly which samples will come from each component
    u = np.random.random((1,N))
    thresholds = np.zeros((1,C+1))
    thresholds[:,0:C] = np.cumsum(priors)
    thresholds[:,C] = 1
    for l in range(C):
        indl = np.where(u <= float(thresholds[:,l]))
```

```
        Nl = len(indl[1])
        labels[indl] = (l+1)*1
        u[indl] = 1.1
        x[:,indl[1]] = np.transpose(np.random.multivariate_normal(meanVectors[:,l], covMatrices[:,:,l],
Nl))

    return x,labels

def plot3(a, b, c, data_set_type, mark="o",col="b"):
#   from matplotlib import pyplot
#   import pylab
#   from mpl_toolkits.mplot3d import Axes3D
#   pylab.ion()
    # fig = pylab.figure()
    #import matplotlib
    #matplotlib.use('Agg')
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(a, b, c,marker=mark,color=col)
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("y")
    if (data_set_type == 'Training'):
        ax.set_title('Training Dataset')
    else:
        ax.set_title('Validation Dataset')
    plt.show()

def predict(data, e):
    return np.matmul(e[:,0], pow(data,3)) + np.matmul(e[:,1], pow(data,2)) + np.matmul(e[:,2],
pow(data,1)) + np.matmul(e[:,3], np.ones((2,1)))

Ntrain = 100
NValidate = 1000
# x_T = traind data X
# y_T = training data y
# x_V = validation data x
# y_V = validation data y
x_T, y_T, x_V, y_V = hw2q2()

# Given = Gamma from 10^(-m) to 10^n
# m = 2 and n = 5
# Generating 100 samples between 0.01 and 100000
gamma = np.linspace(0.01, 100000, 100)
sigma = 0.001
random_noise = np.random.normal(0, sigma)

# Implementation of Maximum-Likelyhood(ML) Estimation
z_T = np.row_stack((pow(x_T, 3), pow(x_T, 2), x_T, np.ones(shape = (x_T.shape[0],
x_T.shape[1]))))
# print(z_T)
```

```
z = z_T.T.reshape(100,4,2)
# print(z)
A = np.zeros(shape = (2,2))
B = np.zeros(shape = (2,4))
# print(A)
# print(B)

# Model Training
for i in range(Ntrain):
   A += np.matmul(z[i,:,:].T, z[i,:,:])
   B += z[i,:,:].T * y_T[i]

# print('product')
# print(z[1,:,:].T * y_T[1])
# print('y_T')
# print(y_T[1])
# print('z')
# print(z[1,:,:].T)
estimate_ML = np.matmul(inv(A), B)

# Model Evaluation using Mean Sqaured Error
Squared_Error = 0
prediction = np.zeros(shape = (NValidate))
for i in range(NValidate):
   prediction[i] = predict(x_V[:, i], estimate_ML)
   Squared_Error += pow((y_V[i] - prediction[i]), 2)

Max_Squared_Error = Squared_Error/1000
print("Maximum Mean Squared Error(MSE) Value is -", Max_Squared_Error)

Max_Squared_Error_array = []
#Implemting MAP for different Gamma values i.e.. from 0.001 to 100000
for i, g in enumerate(gamma):
   for i in range(Ntrain):
      A += np.matmul(z[i,:,:].T, z[i,:,:]) + (sigma / g) * np.eye(2)
      B += z[i,:,:].T * y_T[i]

   estimate_ML = np.matmul(inv(A), B)

   # Model Evaluation using Mean Sqaured Error
   Squared_Error = 0
   prediction = np.zeros(shape = (NValidate))
   for i in range(NValidate):
      prediction[i] = predict(x_V[:, i], estimate_ML)
      Squared_Error += pow((y_V[i] - prediction[i]), 2)

   Max_Squared_Error_array.append(Squared_Error / 1000)

print(f"Minimum Value of MSE is : {np.min(Max_Squared_Error_array)}, Corresponding Gamma
Value is : {gamma[np.argmin(Max_Squared_Error_array)]}")
```

```
print(f"Maximum Value of MSE is : {np.max(Max_Squared_Error_array)}, Corresponding Gamma
Value is : {gamma[np.argmax(Max_Squared_Error_array)]}")

# Visualize variation in Mean Squared Error with respect to gamma
plt.plot(gamma, Max_Squared_Error_array)
plt.xlabel("Gamma")
plt.ylabel("Mean Squared Error")
plt.title("Variation in MSE with respect to Gamma")
plt.show()
```